BRL-MR-3882

FILE COPY

(2)

MEMORANDUM REPORT BRL-MR-3882

BRL

AD-A230 037

A LOGICAL FRAMEWORK FOR DISTRIBUTED DATA

PAUL BROOME
BARBARA BROOME

DTIC
ELECTE
DEC 19 1990
S B D
Co

NOVEMBER 1990

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

U.S. ARMY LABORATORY COMMAND

BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

90 12 18 122

NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute indorsement of any commercial product.

UNCLASSIFIED

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1990	Final 9/1/89 - 8/31/90

4. TITLE AND SUBTITLE

A Logical Framework for Distributed Data

5. FUNDING NUMBERS

1L161102AH43
44592-001-05-3301
RDT&E
44043-010-37
1L162618AH80
44592-002-46-3702

6. AUTHOR(S)

Paul Broome and Barbara Broome

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U. S. Army Ballistic Research Laboratory
ATTN: SLCBR-SE-W
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U. S. Army Ballistic Research Laboratory
Attn: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-5066

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

BRL-MR-3882

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

In this paper we consider logic programming as a means of both computing and formulating complex queries in the same system and describe the application of these concepts to a medium sized database. In particular, we establish a term representation of the data used in an experimental battlefield information system and conceptually extend this database with rules.

We develop browsing operations for that system by combining logical operations with constraints. These programs and queries have mathematical properties that can be specified as equations between relations. We then apply these equations to support program transformations that improve query efficiency. This work increases the likelihood of performing declarative operations on the constantly changing data associated with distributed databases.

14. SUBJECT TERMS

Logic programming, program transformation, relation algebra, distributed database, battlefield management, data management, data bases

15. NUMBER OF PAGES

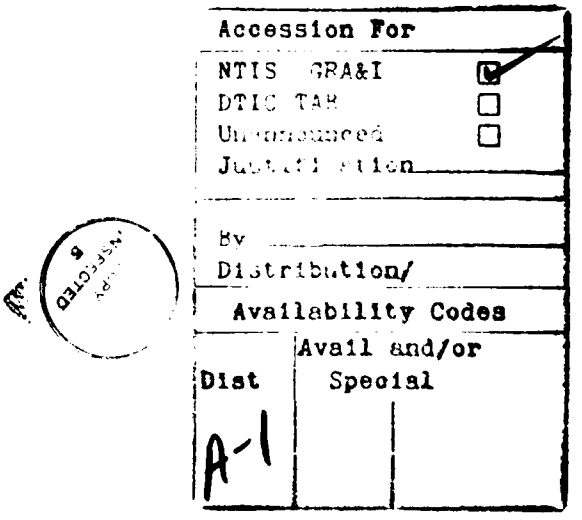
21

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

This page intentionally left blank.

Contents



This page intentionally left blank.

List of Figures

This page intentionally left blank.

Acknowledgements

The authors thank George Hartwig, Eric Heilman and Ken Smith for providing background information on the Information Distribution System, and for their advice regarding FACTBASE query requirements. The authors also thank Morton Hirschberg for his encouragement.

This page intentionally left blank.

1 Introduction

Database management systems have become widely recognized as a means of sharing and maintaining data in a way that avoids redundancy and inconsistency. They allow the user to insert, delete and modify data and perform simple queries with a minimum of effort.

In recent years, however, the use of database systems has been extended to more and more complex applications. Databases address not just the predictable information required by a personnel department of a company, but also the less predictable information required by an object oriented simulation, an expert system, or a battlefield commander. Techniques developed with business applications in mind do not always provide the query flexibility required. Further, they do not extend themselves easily to take advantage of rapidly developing technologies like parallel computation and automatic program transformation.

Logical databases are very attractive for maintaining and manipulating knowledge and are predicted by some to be the data management system of the future[1]. Reasons for this prediction are that the approach is: well founded, as it is based on logic; cohesive, as it allows data structures, queries and computations in a single notation; declarative and therefore non-sequential, providing more potential for tapping the faster computing speeds of parallel processors. These features can greatly improve program maintenance, reliability, generality and efficiency.

In this project we select an existing distributed fact base and reformulate it as a logical database. Next, we construct some sample queries. Finally, we address possible query transformations and their impact on the efficiency of the associated queries. This approach allows evaluation of the logical database approach: the relative ease of development, query flexibility and efficiency. These issues are addressed in this paper. Further, the dynamic nature of the knowledge base selected allows us to examine compromises between absolute logical correctness and conclusions based on imperfect, incomplete, or changing data. Future work will examine this problem, as well as data visualization and query scheduling.

2 The Information Distribution System

Battlefield management has been identified as a major thrust for future Army technological development[2]. Here we find a prime example of the

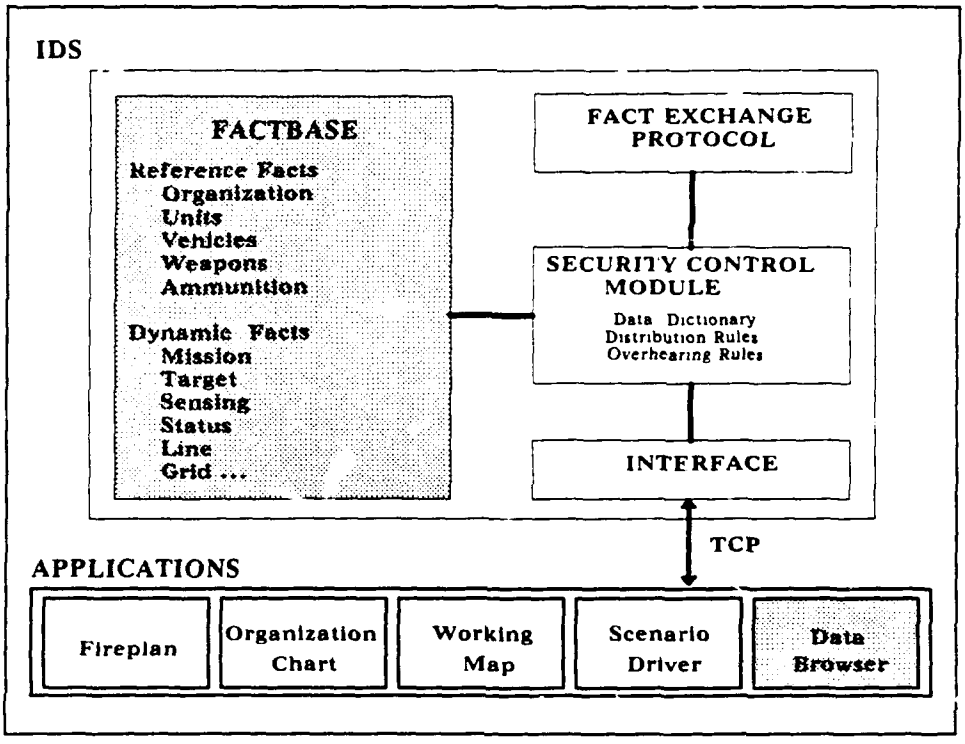


Figure 1: The Information Distribution System. (For this project, browsing operations are being developed to query the FACTBASE.)

need for both query flexibility and efficiency. In a highly dynamic, unpredictable and hostile combat environment, it is crucial that queries be easily formulated and quickly resolved.

The Information Distribution System (IDS) was developed as an experimental prototype to evaluate various data abstraction and distribution technologies for automatically distributing information to and among fighting level forces. It assumes low bandwidth communications in the tactical combat environment. Specifically, it addresses how to insure required battlefield information is available at the various locations where the battlefield management function is performed. As part of this prototype, a FACTBASE was developed, which accommodates the wide variety of information required at brigade and below. Various application programs then access the FACTBASE information through the IDS interface [3]. Figure 1 illustrates the IDS structure and its relationship to the various IDS applications.

The FACTBASE consists of various C programming structures and has a small query language with a C-like syntax. Some facts are relatively static over time, while others are more dynamic [4]. The information in the FACTBASE is complex, requiring all three possible database schemes: hierarchical, for the organizational structure; network, for the communications connectivity; and relational, for the logistics data found in TO&E or equipment manuals. This FACTBASE serves as the foundation for our logical database.

3 Logical Databases

Logic is a branch of mathematics which allows the explicit expression of goals, knowledge, and assumptions. It supplies a foundation for deducing conclusions from premises and for determining validity and consistency. Logic programming is a formal system for specifying objects and relations between objects. It departs radically from the mainstream of computer languages. It is not derived from a physical machine's instruction set, but is instead founded on an abstract model based on first order logic[5]. A logical, or deductive, database is a set of facts that are combined with a set of rules to allow new facts to be inferred and new relationships to be defined. A logical database is firmly and declaratively founded on a small, but powerful, set of primitives. This characteristic increases reliability, confidence, and efficiency.

Some of the dominant areas of interest in logic programming are program correctness, program optimization, parallelism and program synthesis. Major applications of logic programming have been made to intelligent databases, natural language processing, computer aided design, molecular biology, and high level compilation.

Logic programming attempts to apply the rigor of formal logic to complex, computer-based systems that lack such logical foundations. It is an ideal that has not been, and may never be, realized on an existing machine. One approximation is given by the programming language, Prolog. Prolog compilers have become very efficient primarily as a result of work by Warren and his colleagues[6]. This application is being developed in Prolog.

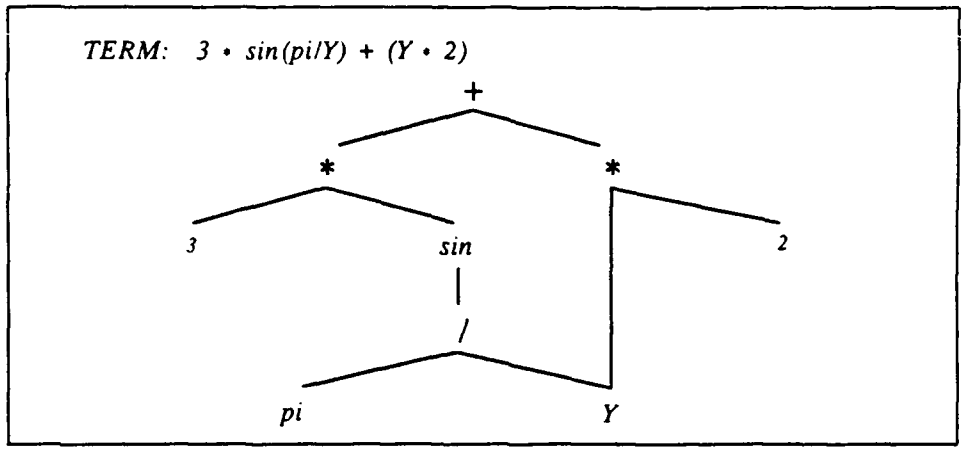


Figure 2: A graphical depiction of a term.

4 Developing a Logical FACTBASE

We began this project by constructing a parser and translator to transform the IDS FACTBASE into equivalent logical relations, which we refer to as the Logical FACTBASE. The result of the translation is a collection of approximately 30,000 Prolog clauses. This representation can include networks, hierarchies and relations. For the initial phase of the project, we have confined ourselves to the static portions of the database, intending to address the dynamic portions in the future. The static portions include the general unit or system properties while the dynamic portions include such changing values as unit location or assignment.

The founding data structure for the database is the term, made up of variables and constants. *Variables* are represented by character strings beginning with an upper case character. Special characters and strings beginning with lower case characters are *constants*. As Figure 2 illustrates, a term may be thought of as a tree-like structure with leaves that are variables or constants (like 3, pi, Y or 2 in Figure 2). The root and internal nodes of the graph are constants and are called *function symbols* ($+, *, sin$ and $/$). The root ($+$) is the *principal function symbol.* It is important to note that function symbols are passive, syntactic objects without any implied interpretation.

More precisely, a *term* is either a variable, a constant, or a function symbol with arguments that are terms. The most general term is simply

a variable. A term whose leaves are all constants is called a *ground term.* In the usual Prolog system, constants are stored only once and all other occurrences are simply pointers to the centrally-stored constant. Similarly, if a variable occurs twice in a term, both occurrences refer to the same variable (like Y in Figure 2). Thus, a term is not really a tree but a directed, acyclic graph, that is, a tree with shared branches. This sharing can mean significant savings in storage and is a side effect of the unification algorithm, discussed in the next section.

One special kind of term is the list. A *list* is made up of a nested sequence of pairs indicated with the period as principal function symbol. For example, a list of the first five integers is $.(1,.(2,.(3,.(4,.(5,[\ ])))))$, where we are representing the empty list with [ ]. More conveniently, we can represent this list as $[1,2,3,4,5]$.

Intuitively, a term may make up an entire fact or it may be the argument in a rule stating a fact. Terms also play the role of arrays, pointers, and record data structures.

A *rule* is the fundamental statement in a logic program or logical database. A rule has a head and body separated by ': −'; it ends with a period. The head contains at most one term, and the body contains zero or more terms separated by a comma. We can read a rule declaratively, that is as a statement of fact. For example,

$$P :- Q, R.$$

means that P is true if Q is true and R is true. A rule is also called a *clause.* A *unit clause* is a clause in which the body is empty. A *logic program* is a set of clauses.

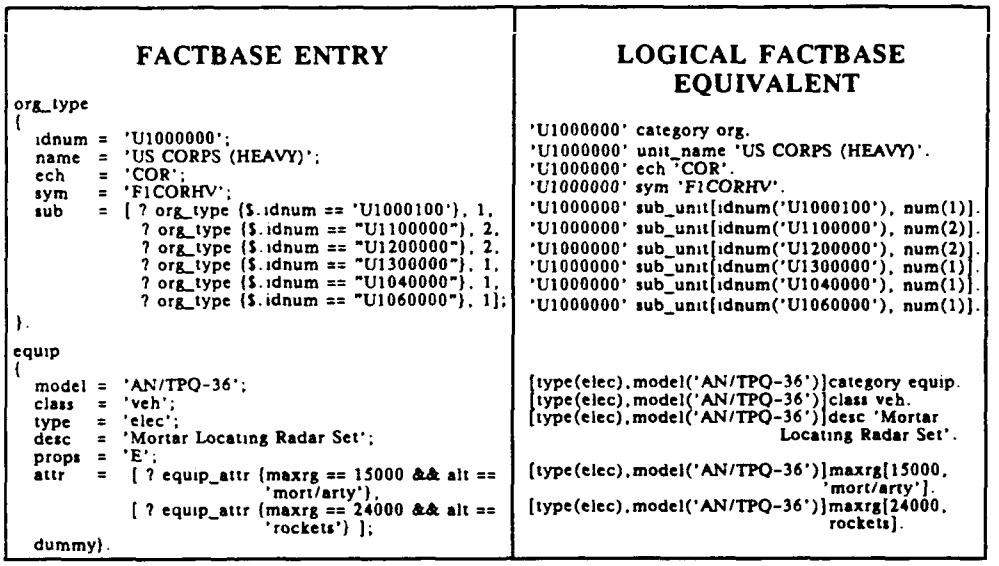
The IDS data was translated into unit clauses whose principal function symbols have two arguments. These define proper binary relations and are to be read as statements of fact. An example would be the clause

$$ech('U1000000','COR').$$

This is a unit clause whose head is a single term. The principle function symbol is *ech* and it has two arguments, *'U1000000'* and *'COR'*. The function symbol can also be placed between its arguments, in *infix* form, as

$$'U1000000'\ ech\ 'COR'.$$

Binary representation was chosen for several reasons. First, it is simple; database entries are easily written, easily searched, and can often be read



FACTBASE ENTRY

```
org_type
{
  idnum  =  'U1000000';
  name   =  'US CORPS (HEAVY)';
  ech    =  'COR';
  sym    =  'F1CORHV';
  sub    =  [ ? org_type ($.idnum == 'U1000100'), 1,
              ? org_type ($.idnum == "U1100000"), 2,
              ? org_type ($.idnum == "U1200000"), 2,
              ? org_type ($.idnum == "U1300000"), 1,
              ? org_type ($.idnum == "U1040000"), 1,
              ? org_type ($.idnum == "U1060000"), 1];
}.

equip
{
  model  =  'AN/TPQ-36';
  class  =  'veh';
  type   =  'elec';
  desc   =  'Mortar Locating Radar Set';
  props  =  'E';
  attr   =  [ ? equip_attr (maxrg == 15000 && alt ==
                            'mort/arty'),
              [ ? equip_attr (maxrg == 24000 && alt ==
                              'rockets') ];
  dummy).
```

LOGICAL FACTBASE EQUIVALENT

```
'U1000000' category org.
'U1000000' unit_name 'US CORPS (HEAVY)'.
'U1000000' ech 'COR'.
'U1000000' sym 'F1CORHV'.
'U1000000' sub_unit[idnum('U1000100'), num(1)].
'U1000000' sub_unit[idnum('U1100000'), num(2)].
'U1000000' sub_unit[idnum('U1200000'), num(2)].
'U1000000' sub_unit[idnum('U1300000'), num(1)].
'U1000000' sub_unit[idnum('U1040000'), num(1)].
'U1000000' sub_unit[idnum('U1060000'), num(1)].


[type(elec),model('AN/TPQ-36')]category equip.
[type(elec),model('AN/TPQ-36')]class veh.
[type(elec),model('AN/TPQ-36')]desc 'Mortar
                                  Locating Radar Set'.

[type(elec),model('AN/TPQ-36')]maxrg[15000,
                                     'mort/arty'].
[type(elec),model('AN/TPQ-36')]maxrg[24000,
                                     rockets].
```

Figure 3: An example of an IDS fact and its translation to proper relation form.

as if they were sentences. Second, with this approach, there is no loss of computational power. Rules on binary relations can compute anything that rules on n-ary relations can compute[7]. Finally, the method we use later for transforming queries requires that the relations have two arguments[8].

Figure 3 illustrates the translation of two FACTBASE entries from their original C structure into their logical representation. The C structures typically consist of a fact type, followed by a series of subfield identifiers which are associated by = with a subfield value. In the example, *org_type* and *equip* are both fact types. Looking more closely at *org_type*, *idnum* is a subfield identifier, and its value is *U1000000*, a unique unit identification code developed for IDS applications. A unit clause is asserted for each of these triples, with the subfield identifier becoming the binary relation. The fact type and subfield value are the relation's arguments. A subfield value of *'E'* indicates an empty field and is not translated. In the example, one organizational fact is translated to 10 unit clauses. Their principal function symbols are *category, unit_name, ech, sym and sub_unit.* Each relation has 2 arguments. The sub_unit function, for example, has 2 arguments: parent unit id; and a list of 2 terms, the subunit and its number of occurrences.

After the translation was accomplished, a small parser was written in

Prolog, in which the operator precedence, position, class and associativity were established. The binary relations resulting from the translation were all defined in infix form.

Finally, the database was extended with new relations. These relations were not part of the organizational or logistical structure, but were created to help form new queries. For example, as illustrated in Figure 3, we know the maximum range of our weapons. We can extend the data by defining what we mean for a given distance, R, to be within firing range of a particular weapon of type T and model M:

$$\begin{array}{ll} [T, M] & can\_fire\_at\_targets\_at\_range\ R :- \\ & [T, M]\ maxrg\ [Range, \_Alt], \\ & Range\ >\ R. \end{array}$$

This new relation could be useful in searching for the right weapon to use against a given target. The new relations extend the translated database entries to a conceptually larger database. They are, in fact, rules that assist in formulating queries. This brings us to our next topic.

5 Querying the Logical FACTBASE

The next step in the application was to construct some queries. The fundamental tools for querying are unification and backward inferencing. We therefore begin this section with a brief explanation of these basic procedures.

The *unification algorithm* is a solution procedure that derives values for variables from an equation between two terms. Given two terms S and T the unification algorithm determines values for variables as follows:

- if S and T are both constants then unification succeeds if they are identical and fails if they are different.
- if S is a variable, then the value for S is $S = T$. (Symmetrically, if T is a variable, then the value for T is $T = S$.)
- if S and T are more general terms with the same function symbols, then the solution is determined by corresponding unification of their arguments.
- if S and T are more general terms with different function symbols then unification fails.

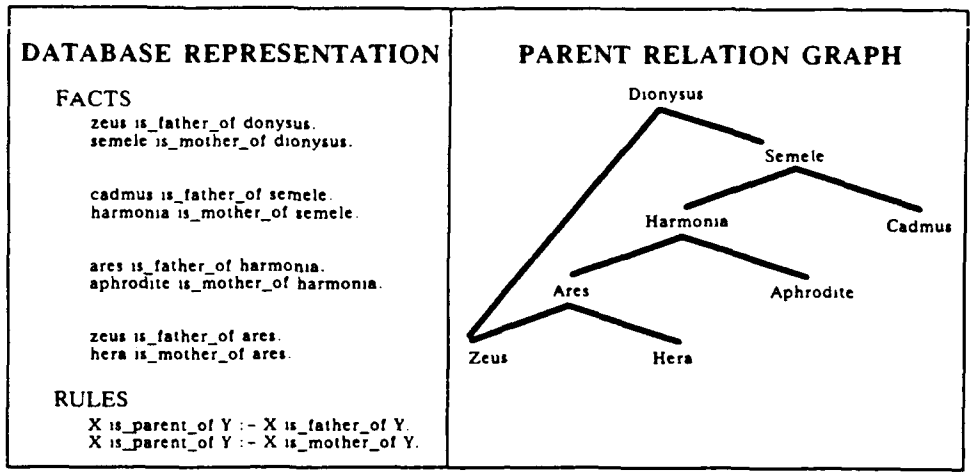


Figure 4: Representing the parent relationship in a logical database.

Unification, then, can be applied to extract components of clauses. Figure 4 illustrates a familiar example of a family database [9]. In this example, consider unifying the two terms X *is_father_of ares* and *zeus is_father_of* Y. From

$$X \; is\_father\_of \; ares = zeus \; is\_father\_of \; Y$$

we would conclude that a value for X is $X = zeus$ and a value for Y is $Y = ares$.

The second fundamental tool is ***backward inferencing***, which is essentially the application of one rule to a goal, reducing it to a conjunction of subgoals. Inferencing allows us to arrive at conclusions from facts and rules. For example, in Figure 4, *zeus is_parent_of* Y can be reduced to *zeus is_father_of* Y using the very first rule allowing us to eventually infer that $Y = dyonysus$. If we look for more solutions, we find that $Y = ares$ also satisfies the query.

A *goal*, or in our case a database query, is a clause with an empty head. This goal is a conjunction of subgoals which is solved by solving all subgoals. Each subgoal is solved by unifying it with the head of a clause in the database. This creates values for variables. A single backward inference reduces this subgoal to another conjunction of subgoals until reaching the subgoal *true*, which is trivially solvable. In Prolog, subgoals are solved in sequential, left to right order and clauses are chosen in top to bottom order

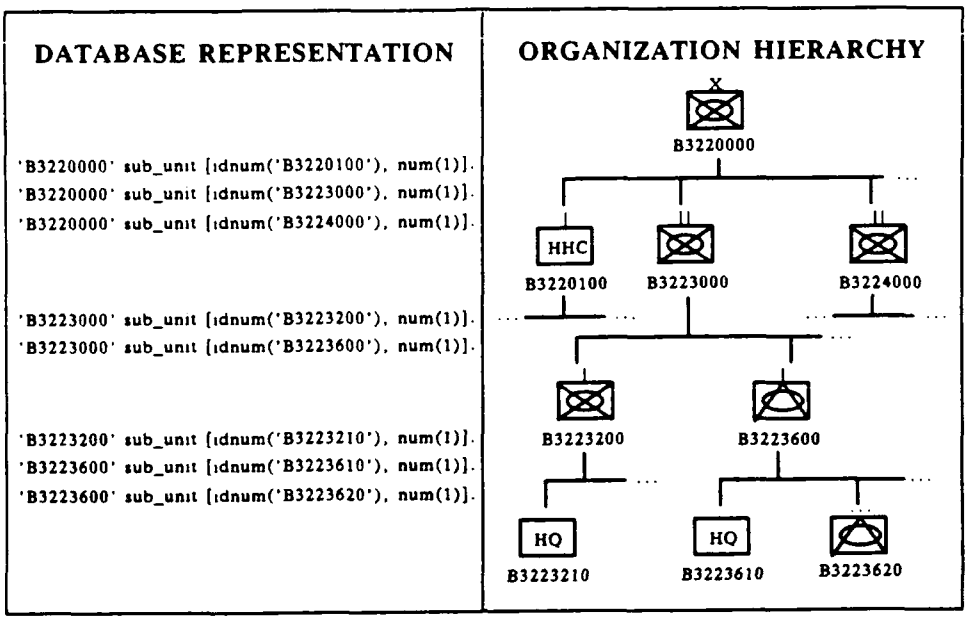


Figure 5: Representing the subunit relationship in the Logical FACTBASE.

with backtracking to find additional solutions. Again, looking at Figure 4, we can determine who are the parents of Semele by solving the goal

$$:-X\ is\_parent\_of\ semele.$$

This unifies with the head of the first rule, yielding $X\ is\_father\_of\ semele$. The solution for X in that subgoal is $X = cadmus$. Alternatively, the goal resolves to $X\ is\_mother\_of\ semele$, in which we find a alternate solution $X = harmonia$.

In Figure 5 we extend this technique to the FACTBASE data, using the subunit relation somewhat like the parent relation. *A subunit B* means that A and B are members of the *subunit* relation, with A being the parent unit.

Once the database has been established, a number of queries can be solved without any programming, by the selective placement of constants and variables in goals. Prolog attempts to unify the goal with unit clauses in the database. For example, using the data in Figure 5, we may identify all the subunits of B3220000, with the simple query, *'B3220000' sub_unit X*. Further, all relations defined with unit clauses can be queried in either

direction. This is a powerful aspect of the unification algorithm, for it allows us to answer questions about the converse of a relation in the database as well as about the relation itself. For example, the subunit relation has been defined, so we have immediate access to its converse, the parent relation. That is, we can identify the parent unit for B3223600 through the query, *X sub_unit [idnum('B3223600'), _num].* Similar queries can be made for all relations established in the database. Queries solved with a single unification are satisfied almost immediately.

As indicated previously, more complex queries may require the definition of new relations. Suppose we wish to know whether B3223610 is under the control of B3223000. In this case, we would like to know if B3223610 is a subunit of B3223000, or if it is a subunit of a subunit of B3223000, etc. We define the *controls* relation recursively as follows:

$$\begin{aligned} A \textit{ controls } B :- \quad & A \textit{ subunit } B. \\ A \textit{ controls } B :- \quad & A \textit{ subunit } C, \\ & C \textit{ controls } B. \end{aligned}$$

Now, we may query with the goal *'B3223000' controls B3223610'*. Prolog verifies that there is a path through the organization graph in Figure 5 from B3223000 to B3223610 through B3223600, returning the answer *true.*

6 Query Transformations

Finally, we address possible query transformations and their resulting impact on the efficiency of the associated queries. Sometimes the most obvious expression of a query is not the most efficient for implementation, as illustrated in the example below. One of the benefits we hope to derive from this logical approach to computation is to be able to state queries in a straightforward manner, and then reliably transform these queries to optimize their execution.

The solution procedure for a query starts by unifying the goal with the head of a clause to determine values for variables. This environment is used to solve each subgoal of the body in turn. If any subgoal is unsolvable then alternate clauses are applied by backtracking to create possible alternate paths. A solution can be found more efficiently if the search can be correctly constrained in the appropriate direction. But note that an overconstrained system may be unsolvable.

Consider the problem of searching for a path through a graph described by a relation R. This is essentially asking if the two endpoints $\langle x, y \rangle$ of the

graph are members of the transitive closure R^+ of R. A pair is a member of the transitive closure of R if either the pair is in R or there is an intermediate point z such that $\langle x, z\rangle$ is in R and $\langle z, y\rangle$ is in R^+. In symbols this is written as

$$R^+ = \{\langle x, y\rangle | \langle x, y\rangle \in R \text{ or } \exists z, \langle x, z\rangle \in R, \text{ and } \langle z, y\rangle \in R^+\}.$$

Operationally, R^+ is the exhaustively repeated application of R.

The *controls* relation, that is the transitive closure of the *subunit* relation, provides a perfect example of how we can improve the efficiency of the solution procedure by transforming the query. In this example, we say that *A controls B* if there is a path from A to B in the graph formed by the *subunit* relation. The *controls* definition naturally schedules subgoals from the top of the command hierarchy downward. As illustrated in the following example, this schedule is inappropriate and inefficient for the database as structured. A bottom up search would have been better.

Consider the command hierarchy depicted in Figure 6. In this graph, the lines indicate the *subunit* relation, with higher nodes indicating parent units and lower nodes their subunits. This simplified example allows us to limit the *controls* relation to two levels. That is, a unit *controls* its subunits and its subunits' subunits. To determine if *B*3224600 is under the control of *B*3220000 we find an intermediate unit *V* such that *B*3220000 *subunit V* and *V subunit B*3224600. Efficiency greatly depends on which subgoal is selected first. If we start with the goal *B*3220000 *subunit V* then we have multiple solutions, requiring us to travel through the tree, first through node B3220100 and its subunits, then through node B3223000 and its subunits, and finally to our solution point under B3224000. On the other hand, if we start with the goal *V subunit B*3224600, it has a unique solution, quickly generating our solution path.

The reason that the second subgoal should be chosen first is that the converse of the *subunit* relation, denoted ($subunit^\smile$), is a function. Each unit has exactly one parent unit. Thus it would be much more efficient to carry out the search in this order, as each choice would be unique. We, therefore, transform the query to find a path in the tree with

$$controls^\smile = (subunit^\smile)^+,$$

denoting transitive closure with $^+$. The *subunit* relation does indeed define a tree, so *A controls B* is reversible. Since the converse of *subunit* is a function, the paths through the tree can be most efficiently found by

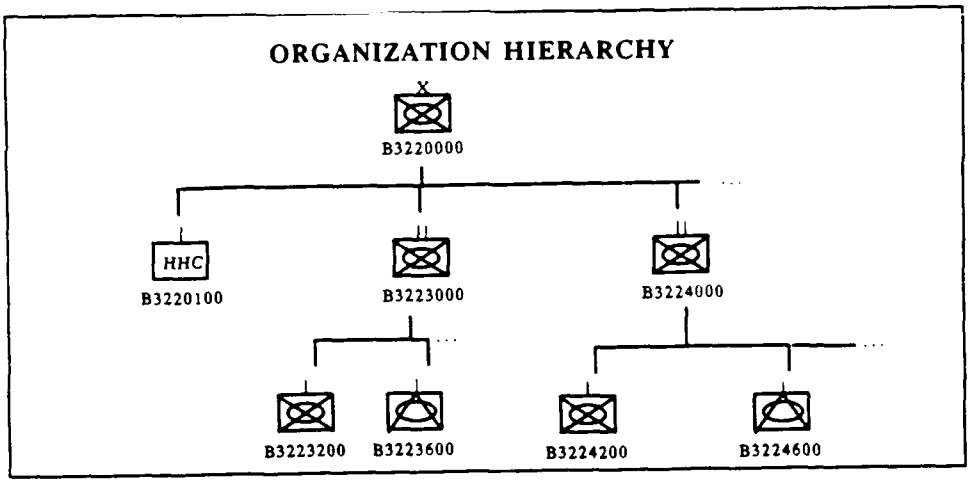


Figure 6: A sample command hierarchy graph.

searching up the tree instead of top down. The bottom up search requires no backtracking. Here we note that it is the nature of the *subunit* relation that suggests this transformation. For the large organizational structure in the IDS, the bottom up solution of a sample query was immediately solved whereas the corresponding top down query took more than an hour.

The reversibility of the unification algorithm is what allows us to represent converse relations. Some knowledge about reversibility can save a great deal of computation time. Searches both up and down the hierarchy in the originally defined IDS database would have required that we add the converse relations to the data, essentially doubling the storage requirements for the *subunit* relation. This trades storage for time, and sacrifices modularity and maintainability. With our new approach, the tradeoff is unnecessary.

On the other hand, while queries are completely reversible when solved with unit clauses, termination is unpredictable in general. In Prolog, some queries that can be easily solved in the forward direction may not terminate in the reverse direction. In addition, some operations in Prolog only have meaning when all arguments are ground terms. Attractive solutions to these problems are emerging from research in constraint logic programming and higher order extensions to logic programming[8,10]. These approaches solve bigger classes of problems by giving declarative extensions to some operations in logic programming such as negation, inequality, and ordering.

7 Future Work

Future work will emphasize three main areas: first, the notoriously difficult problem of synchronizing data updates with data queries, including determining constraints that can maintain integrity; second, methods of pictorially representing the relations in the Logical FACTBASE and the associated queries; and, finally, further query optimizations.

In Section 2 we indicated that the static portions of the FACTBASE were translated first. The dynamic data would be translated in future. This is because logic programming with a set of clauses does not accommodate axioms that may be modified in the middle of a deduction [11]. An attractive compromise, however, can be derived from a thorough treatment of binary relations[8,12]. Accepting the fact that change is an integral part of our distributed database, we concentrate on cleanly separating the abstract portions of our relations, the rules, from the facts. That is, we separate the program from the data. Once this is accomplished, the algebra of equations between relations is an appropriate formalism and an ideal foundation for query optimizations that hold independently of the data. The FACTBASE information will be set aside as an area designated to be modified. Queries operate on a snapshot of the database without attempting to maintain a notion of logical truth. Equations between combinations of relations hold independently of the data. We extend this concept and further partition the data into distinct relations to represent partitions of the database such as *subunit* and *owns_equipment*. Then we can pass these relations along as arguments to the previous operations. This adds another level of generality to the query language so that generic operations can be defined and applied to portions of the database or to other predefined operations on the database.

Secondly, we will experiment with ways of pictorially representing the relations in the logical FACTBASE and the associated queries. There is a close relationship between proper binary relations and combinatorial graphs. This strongly suggests a visualization technique for logical databases that may allow the casual user to bypass much of the notation and abstract syntax.

Finally, we will explore schedules for constraints as binary relations. This includes further methods for reordering subgoals, merging recursions, and propagating constraints. There is also a close relationship between declarative languages and parallelism. The mathematical properties of program operations such as associativity and commutativity indicate that order of some computations can be ignored.

8 Conclusions

We selected an existing distributed fact base and reformulated the static portion as a logical database of binary relations. A parser of C structures was built and a translator constructed to separate the information into relations for querying and updating. We identified the operations required to develop our queries. Finally, some high level, decision critical queries were formulated to test flexibility. Simple query transformations were applied to improve efficiency.

At the end of this first phase, we find that the logical database has a relatively simple structure. Once its structure was established, a number of queries were immediately available through unification. These were satisfied almost instantaneously. More complex queries were built using rules as statements of a recursive programming language, with power, flexibility and limited reversibility. The approach to date puts us in a position to begin examining the effort required to develop queries and the computation time required to perform those queries on the data one might expect in a *battlefield environment*.

A single inference is comparable to one statement executed in a procedural language. The number of inferences involved is critical to efficiency and may be very large if the order of subgoal selections is not carefully controlled. Prolog queries are not always reversible, partly because subgoals are chosen in a predetermined order. This makes naive queries more difficult to formulate and implies that careful attention must be paid to the solution procedure when scheduling subgoals. A view of programs as proper binary relations, along with an associated set of equations between relations, is a step toward understanding and harnessing the limited reversibility of logic programs.

The primary claim of this work is that logical databases are a convenient vehicle for the management of battlefield information. The primary advantages are improved program maintenance, reliability, efficiency and generality. While no system can perfectly represent a distributed database, we have begun applying a logical model that is an attractive compromise, viewing both queries and data as proper binary relations. The query language we will use has a set of operations with an associated theory. This theory is independent of the data and should be unaffected by its volatility.

References

[1] J. W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, Berlin, 1987.

[2] Science Applications International Corporation , "The Army Technology Base Master Plan," SAIC-MCDC 89-03002, April 1989.

[3] Samuel C. Chamberlain, "The Information Distribution System: IDS - An Overview," Ballistic Research Laboratory, BRL-TR-3114, Aberdeen Proving Ground, MD, August 1990.

[4] George W. Hartwig, *The Information Distribution System: The Factbase*, Ballistic Research Laboratory, Aberdeen Proving Ground, MD, (To be published).

[5] L. Sterling & E. Shapiro, *The Art of Prolog: Advanced Programming Techniques*, MIT Press, Cambridge, 1986.

[6] D.H.D. Warren, "An Abstract PROLOG Instruction Set," SRI International Technical Note 306, 1983.

[7] J. Sebelik and P. Stepanek, "Horn Clause Programs for Recursive Functions," in *Logic Programming*, K. L. Clark and S.-A. Tarnland, ed., Academic Press, London, 1982.

[8] Broome, Paul, "Program Transformation with Abstract Relation Algebras," BRL-MR-3784, October 1989.

[9] R. Kowalski, *Logic for Problem Solving*, North Holland, New York, 1979.

[10] J. Cohen, "Constraint Logic Programming Languages," in *Communications of the ACM* #33, July, 1990.

[11] J. McCarthy & P.J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in *Machine Intelligence*, D. Michie and B. Meltzer, ed. #4, Edinburgh University Press, Edinburgh, 1969.

[12] A. Tarski and S. Givant, "A Formalization of Set Theory Without Variables," in *Colloquium Publications, American Mathematical Society* #41, Providence, RI, 1987.

This page intentionally left blank.

No of Copies	Organization
2	Administrator Defense Technical Info Center ATTN: DTIC-DDA Cameron Station Alexandria, VA 22304-6145
1	HQDA (SARD-TR) WASH DC 20310-0001
1	Commander US Army Materiel Command ATTN: AMCDRA-ST 5001 Eisenhower Avenue Alexandria, VA 22333-0001
1	Commander US Army Laboratory Command ATTN: AMSLC-DL Adelphi, MD 20783-1145
2	Commander US Army, ARDEC ATTN: SMCAR-IMI-I Picatinny Arsenal, NJ 07806-5000
2	Commander US Army, ARDEC ATTN: SMCAR-TDC Picatinny Arsenal, NJ 07806-5000
1	Director Benet Weapons Laboratory US Army, ARDEC ATTN: SMCAR-CCB-TL Watervliet, NY 12189-4050
1	Commander US Army Armament, Munitions and Chemical Command ATTN: SMCAR-ESP-L Rock Island, IL 61299-5000
1	Commander US Army Aviation Systems Command ATTN: AMSAV-DACL 4300 Goodfellow Blvd. St. Louis, MO 63120-1798

No of Copies	Organization
1	Director US Army Aviation Research and Technology Activity ATTN: SAVRT-R (Library) M/S 219-3 Ames Research Center Moffett Field, CA 94035-1000
1	Commander US Army Missile Command ATTN: AMSMI-RD-CS-R (DOC) Redstone Arsenal, AL 35898-5010
1	Commander US Army Tank-Automotive Command ATTN: AMSTA-TSL (Technical Library) Warren, MI 48397-5000
1	Director US Army TRADOC Analysis Command ATTN: ATRC-WSR White Sands Missile Range, NM 88002-5502
(Class. only) 1	Commandant US Army Infantry School ATTN: ATSH-CD (Security Mgr.) Fort Benning, GA 31905-5660
(Unclass. only) 1	Commandant US Army Infantry School ATTN: ATSH-CD-CSO-OR Fort Benning, GA 31905-5660
1	Air Force Armament Laboratory ATTN: AFATL/DLODL Eglin AFB, FL 32542-5000
	Aberdeen Proving Ground
2	Dir, USAMSAA ATTN: AMXSY-D AMXSY-MP, H. Cohen
1	Cdr, USATECOM ATTN: AMSTE-TD
3	Cdr, CRDEC, AMCCOM ATTN: SMCCR-RSP-A SMCCR-MU SMCCR-MSI
1	Dir, VLAMO ATTN: AMSLC-VL-D

No. of Copies	Organization
1	Director US Army Research Office ATTN: Dr. William Sander P.O. Box 1211 Research Triangle Park, NC 27709-2211
1	Director US Army Research Office ATTN: Dr. Jerry Andersen P.O. Box 1211 Research Triangle Park, NC 27709-2211
1	Director US Army Research Office ATTN: Dr. Jagdish Chandra P.O. Box 1211 Research Triangle Park, NC 27709-2211
1	PM-CHS ATTN: SSAW-CC-CHS, Mr. S.H. Levine Fort Monmouth, NJ 07703-5000
1	PM-FATDS ATTN: AMCPM-TF, Mr. Jack Piper Bldg 457 Fort Monmouth, NJ 07703-5027
1	PEO-Command & Control Systems ATTN: SPIS-CC, Mr. Giordano Fort Monmouth, NJ 07703-5000
1	Project Manager TACFIRE/FATDS ATTN: AMCPM-TF Fort Monmouth, NJ 07703-5022
1	Commander TRAC-WSMR ATTN: ATRC-WFB, Dr. Deason WSMR, NM 88002-5502
1	Director, Advanced Technology Joint Tactical Fusion Program Office ATTN: SFAE-JT-AT, Dr. Ray Freeman McLean, VA 22102-5099

No. of Copies	Organization
1	Technical Director Joint Tactical Fusion Program Office ATTN: SFAE-JT-AT, Mr. Millard Hunter McLean, VA 22102-5099
1	GE ATCCS SE&I ATTN: Mr. Neil Verstermark Building 97, P.O. Box 8048 Philadelphia, PA 19101
1	Cornell University Mathematical Sciences Institute ATTN: Prof. Anil Nerode 294 Caldwell Hall Ithaca, NY 14853-26022
1	University of California Department of Computer Science ATTN: Dr. Michael Stonebraker Berkeley, CA 94704
1	Magnavox ATTN: Mr. M. Meier 1313 Production Road Fort Wayne, IN 46808
	Aberdeen Proving Ground
1	Dir, VLAMO ATTN: AMSLC-VL-CB, Mr. W. Hughes

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. BRL Report Number BRL-MR-3882 Date of Report NOVEMBER 1990

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

Name

Organization

Address

City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the New or Correct Address in Block 6 above and the Old or Incorrect address below.

OLD ADDRESS

Name

Organization

Address

City, State, Zip Code

(Remove this sheet, fold as indicated, staple or tape closed, and mail.)

---FOLD HERE---

DEPARTMENT OF THE ARMY
Director
U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-5066
OFFICIAL BUSINESS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT No 0001, APG, MD

POSTAGE WILL BE PAID BY ADDRESSEE

Director
U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-9989

---FOLD HERE---